C85/C95/C105

Projector Control

REVISED	VERIFIED	APPROVED
	n.a.	n.a.

1 SETTING UP

To enable serial control, do the following:

- Remove the check mark from "**Serial Mouse"** in the Projector's FEATURE MENU. This changes the Baudrate from 1200 to 9600 and turns of the Serial mouse data.
- Connect the serial (COM) port from the Host to the RS232 connector on the projector

1.1 Rs232 Control Cables

Please order the following parts:

Part Number	Description
301.102A	Mouse Cable 4 meter
301.113B	PC Mouse Adapter RS232 Straight

Table 1

1.2 RS232 Settings

If SerialMouse is deselected the setting is:

Parameter	Default setting
Baudrate	9600
Data bit	8
Parity bit	None
Stop bit	1
Flow Control	None

Table 2

If SerialMouse is selected the setting is:

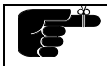
Parameter	Default setting
Baudrate	1200
Data bit	8
Parity bit	None
Stop bit	1
Flow Control	None

Table 3

	If the serial mouse is enabled the projector sends mouse data whenever the user moves the trackball on the Batmouse.

2 GENERAL

This document describes the Communication Protocol between a Proxima® Projector and a Computer (Host). By using this connection the Host will be able control the projector.

	This document is HEX command based only and a regular Terminal program like hyperterminal etc. can therefore not directly be used unless the COM port are activated by the user. This document is aimed at control systems in general which is HEX based.

3 FUNCTION TYPE

The Function is grouped in four different function types:

Function type	Comments
Execute	An *execute* function executes an action on the projector, i.e. only one state.
State	A *State* function performs a set operation from a predefined list of states. Not all values in the range have to be legal.
State String	As state but returns ASCII strings
Adjust	An *adjust* function is characterized by Maximum > Minimum. And all the integer values between Minimum and Maximum are legal.

Table 4

4 PROTOCOL MESSAGES

4.1 Message Formats

The message is always divided in a header and a body part:

Message Head (7Byte)	Message Body (6Byte)

4.1.1 Message Head

The message head has the following structure:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Magic Number			**Body size**		**CRC**	
0xBE	0xEF	0x80	0x06	0x00	CRC_lo	CRC_hi

Table 5

All values is in hexadecimal indicated with the leading 0x. The **Magic number** is used for synchronizing the start of a message. The **Body size** is set to the number of bytes contained in the Body part of the message.

Currently there is only one type of body so the size is fixed to 0x0006. The **CRC** field contains the CRC of the message. The CRC algorithm is given in Appendix 1. To disable CRC set CRC_lo=0x00 and CRC_hi = 0x00.
The Message Head – if the CRC are disabled – then looks like the following string:
0xBE 0xEF 0x80 0x06 0x00 0x00 0x00. This Head will be present in all strings sent to the projector from the host.

4.1.2 Message Body

The message body has the following structure:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Operation Type		**Function**		**Value**	
OpTypeLo	0x00	FuncNum_lo	0x00	Opval_lo	Opval_hi

Table 6

4.1.2.1 Field description

Field Name	Field Value	Description
Operation Type	OpTypeLo	
	0x01	**Operation Set**. Set current value
	0x02	**Operation Get**. Gets the current value
	0x03	**Operation Initialize**. Initialize the value to default. Should not be used
	0x04	**Operation Increment**. Increment the value by one step. Only valid for "Adjust" functions.
	0x05	**Operation Decrement**. Decrement the value by one step. Only valid for "Adjust" functions.
	0x06	**Operation Execute**. Execute the operation. Only used for "Execute" operations
Function	FuncNum	The actual function number to execute. A list of usable FuncNum is given in Chapter 9
Value	-	Only used by Operation Set. The actual value to set the operation to. For all other operation actions, this field is set to 0x0000.

Table 7

5 COMMAND RETURN CODES

The Serial protocol returns an acknowledgment ("return code") for every packet sent. If the host sends a packet, then the projector acknowledge the receipt of this packet. The return codes are detailed in the table below.

Command Return Code Name	Return Code Value	Description
ACK	0x06	Packet acknowledged – no errors. Normal response when receiving
NAK	0x15	Packet not acknowledged – some error occurred in receiving this packet. Usually Indicates a CRC error.
ERR	0x1C 0xXXXX	Packet was received OK, but an error occurred when executing the command contained in the Packet. The two bytes following the 0x1C (0xXXXX) are the error code
VAL	0x1D 0xXX 0xYY	Packet value returned. This return code will be sent in response to a Operation packet of type OPERATION_GET. The first byte after 0x1D (0xXX), is the least-byte of the returned value. The 0xYY is the most significant-byte of the returned value.
STRING	Strings 0x06	ASCII Strings and at last ACK

Table 8

TRANSACTION EXAMPLES

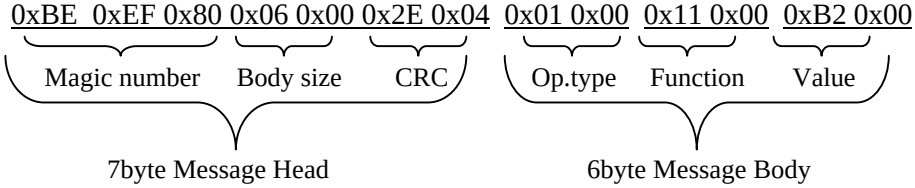
Example 1: Set
Host: Projector
Operation Set Packet =>
<= ACK

Example 2: Get
Operation Get Packet =>
<= VAL

VAL=0x1D 0x00 0x03 means that 0x0300 is returned

Example 3: Hex sequence for Set Brightness to 70% => 0.7*255 = 178 = 0xB2

Host:



Projector:
0x06

Example 4: Hex sequence for power on (with CRC deactivated)

Host:
0xBE 0xEF 0x80 0x06 0x00 0x00 0x00 0x01 0x00 0x01 0x00 0x01 0x00

Example 5: Hex sequence for get Brightness

Host: =>
0xBE 0xEF 0x80 0x06 0x00 0xBD 0x70 0x02 0x00 0x11 0x00 0x00 0x00

<= Projector (70%)
0x1D 0x00 0xB2

Example 5: To get ASCII Information Set Parameter in the State String function

Host: Projector
Operation Set Packet =>
<=ASCII Strings
<= ACK

6 TABLE OF FUNCTIONS

Scr.dep = Source depended

Function	FuncNum	F.type	States	Minimum	Maximum
Power	0x01	State	1 (On) 0 (Off)	-	-
Reset	0x02	Execute	-	-	-
Tune	0x03	Adjust	-	0	255
Width	0x04	Adjust	-	0	4095
Horizontal position	0x05	Adjust	-	0	4095
Vertical position	0x06	Adjust	-	0	4095
ColorTemp	0x07	Adjust	-	0	3
Rear	0x08	State	1 (On) 0 (Off)	-	-
Ceiling	0x09	State	1 (On) 0 (Off)	-	-
Factory reset	0x0A	Execute	-	-	-

Table 9

Function	FuncNum	F.type	States	Minimum	Maximum
Contrast	0x10	Adjust	-	0	255
Brightness	0x11	Adjust	-	0	255
Tint	0x12	Adjust	-	0	255
Color	0x13	Adjust	-	0	255
Sharpness	0x14	State	5(Softest) 6(Soft) 7(Medium) 8(Sharp) 9(Sharpest)	-	-
Keystone	0x15	Adjust	-	0	255
16:9	0x16	State	1(On) 0 (Off)	-	-

Table 10

Function	FuncNum	F.type	States	Minimum	Maximum
Volume	0x20	Adjust	-	0	100
Mute	0x21	State	1 (On) 0 (Off)	-	-

Table 11

Function	FuncNum	F.type	States	Minimum	Maximum
On screen display	0x30	State	1 (On) 0 (Off)	-	-
Source	0x32	State	0 (VGA1) 1 (DVI1) 2 (S-VIDEO1) 3 (CVBS1) 4 (VGA2) 5 (DVI2) 6 (S-VIDEO2) 7 (CVBS2)	-	-
Language	0x33	State	0 (English) 1(Japanese) 2(Norwegian) 3 (Deutch) 4 (Italian) 5 (Spanish) 6 (Korean) 7 (Simpl Chi) 8 (Trad Chi) 9 (Portugeese) 10(French)	-	-
Black	0x34	State	1 (On) 0 (Off)	-	-
Magnify	0x35	Adjust	-	0	26
Pan horizontal	0x36	Adjust	-	Scr.dep	Scr.dep
Pan vertical	0x37	Adjust	-	Scr.dep	Scr.dep
Freeze	0x38	State	1 (On) 0 (Off)	0	1
PnP	0x39	State	1 (On) 0 (Off)	-	-
DPMS	0x3A	State	1 (On) 0 (Off)	-	-
Source Search	0x3B	State	1 (On) 0 (Off)	-	-
SerialMouse[1]	0x3C	State	1 (On) 0 (Off)	-	-
MenuTimeOut	0x3D	Adjust	-	5	50

Table 12

Function	FuncNum	F.type	States	Minimum	Maximum
Information	0x40	State Strings	0 Respons is Source. information . 1 Respons is Service. information	-	-

Table 13

APPENDIX 1: CRC CALCULATION ALGORITHM

The following 'C' code can be used to calculate the 16-bit CRC required for all packets. The CRC is contained in the packet header and is calculated for the entire packet (header plus body). The CRC calculation is performed with the CRC bytes of the packet header initialized to zero.

```
// Using two 256 byte lookup tables, quickly calculate a
16-bit CRC on // a block of data.
// Params:
// pcData : Pointer to data to calculate CRC on.
// nCount : Number of data bytes.
// Return: 16-bit CRC value.
WORD CalculateCRC16 (BYTE *pcData, int nCount)
{
      BYTE cCRCHi = 0xFF; // high byte of CRC initialized
      BYTE cCRCLo = 0xFF; // low byte of CRC initialized
      BYTE cIndex; // will index into CRC lookup table
      while (nCount--) // step through each byte of data
      {
            cIndex = cCRCHi ^ *pcData++; //
            calculate the CRC
            cCRCHi = cCRCLo ^ cCRCHiArray[cIndex];
            cCRCLo = cCRCLoArray[cIndex];
      }
      return (cCRCHi << 8) + cCRCLo;
}
```

```
// Lookup table used for hi-byte of CRC
static const BYTE cCRCHiArray[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00,0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80,0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00,0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00,0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01,0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01,0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00,0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80,0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
};
```

```
// Lookup table used for low-byte of CRC
static const BYTE cCRCLoArray[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6,0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D,0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9,0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA,0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14,0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13,0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33,0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34,0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA,0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9,0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D,0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6,0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0,0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7,0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF,0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8,0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE,0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75,0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1,0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52,0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C,0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B,0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B,0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C,
0x8C,0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86,
0x82,0x42,0x43, 0x83, 0x41, 0x81, 0x80, 0x40
};
```